UNIVERSITY OF GLASGOW

Computing Department

MULTUM OPERATING SYSTEM MEMO NO. 6
(March 1973)

REVISED OUTLINE SPECIFICATION

Preface

This document gives an overview of the concepts and facilities which it is hoped to include in SECANT, an operating system for the MULTUM. These topics are approached from the viewpoint of a user of the system and not from the implementor's angle. It is therefore appropriate to mention here that most of this software will be written in Pascal, the remarkable language of N. Wirth.

The Virtual I/O System, only briefly described herein, is treated more fully in Reference 5. Reference 6, describing the ALP2/3 Executive is essential collateral reading.

Note that the MULTUM Operating System Memo No. 1 is rendered obsolete by the present paper.

C O N T E N T S

1. General Principles.

1.1 Reliability.

It is of paramount importance that the operating system (OS) should fail very infrequently, and that when it does the effects should be confined as closely as possible to the failing component. If this component cannot recover it must be possible to excise it and provide a continuing (albeit degraded) service.

1.2 Generality.

The OS must not exclude potential applications because policy decisions have been inextricably woven into its logic. In particular, it should be possible to accommodate any feasible mix of applications from across the batch - multiaccess - realtime spectrum.

1.3 Minimality.

The OS must be based on the sound implementation of a minimal number of "primitive" facilities.

1.4 Extensibility.

It must be possible to add new facilities to the system by building on the "primitives" and all previously-added facilities.

1.5 Configurability.

It must be possible to configure the OS (a) at system generation time and (b) during use. System generation adapts the system to the hardware of a given installation and to the software policy of its management. Dynamic configuration adapts the system to changes arising during the run, including hardware failures.

1.6 Flexibility.

Policies should be decided upon by the OS, as far as possible, taking into account (a) the current situation, (b) driver tables defining general strategies and (c) installation settable parameters influencing tactics. Where more radical changes of policy are necessary, it should be easy to replace complete modules of the system by modules tailored to the requirements of a particular application.

1.7/

1.7 Modularity.

The implementation should be composed of a set of functional modules. That is the division into modules should be made on the basis of each module's function in the total system rather than on arbitrary rules concerning the size of a module, or any other such consideration.

1.8 Levels of Abstraction.

The modules should be organised hierarchically, with each stage in the hierarchy corresponding to a level of abstraction in the definition of the total system.

1.9 Minimum Privilege.

A component of the system should have the minimum capabilities necessary for it to function.

1.10 Cooperating Sequential Processes.

The system should be implemented as a community of co-operating sequential processes, communicating by means of messages sent to and received from message channels. A process can send a message, wait for a message, receive a message, send a reply, wait for a reply and receive a reply. It can also create and destroy sub-processes, start sub-processes and stop them. (A general semaphore is equivalent to a channel holding only "null" messages.)

A process can be either ready, running or blocked. It is blocked while waiting for some event (e.g. the end of an I/O transfer). It is running if it is being executed in processor. It is ready if it is not blocked and not running. The number of running processes is less than or equal to the number of processors in the configuration. In a sense, a process is a "virtual processor".

This organisation of the OS permits us to take full advantage of the MULTUM multi-processor architecture, so long as it is possible to consider all CPUs as having equal status. (A master/slaves relationship is less satisfactory from the viewpoints of modularity and reliability.)

1.11 Security Spheres.

Each process operates in a "security sphere" defined by the privileges and resources allocated to it. A subprocess necessarily operates in a security sphere wholly contained within that of its creator; that is, a subprocess may have allocated to it only a subset of the privileges and resources of its creator. (Since the security sphere of a process is defined only when the process is created, its creator can make the allocation of resources to it dependent on prevailing conditions. For example in the multiaccess subsystem, the Supervisor might withhold the right to execute user programs interactively during periods of peak load by omitting this privilege from the security sphere of the job organiser process it creates to service the new user's requests. It must also be possible for a process to determine what does lie within its security sphere so that it can avoid any inadvertent transgressions.)

2. The Architecture of the OS.

2.1 The Role of the Executive.

The Executive is responsible for fine control of the haredware and software. Its duties include:

(a) interrupt and trap servicing,

(b) I/O handling at the physical level (definitely) and the logical level (probably).

(c) implementation of the process and message primitives,

(d) construction and maintenance of protective "firewalls" between processes, and,

(e) control of multiprogramming (probably according to a simple priority scheme).

Hardware interrupts and software traps are dealt with by routines in the Executive which we will call handlers. Handlers will normally work almost entirely within special state or in privileged normal state. For this reason they should be as brief as possible.

Realtime functions with highly critical crisis times may be entirely implemented in the appropriate handlers. Less critical functions are initiated by the appropriate handlers but are completed in more leisurely fashion by a process which the handler initiates, or to which it sends a message.

The handlers which implement the numerous Executive software services and primitives are invoked by appropriate Executive call operations which are available to all processes. To implement the principle of "least privilege" it is necessary to associate a set of flags with each process, one flag corresponding to each class of Executive service. The Executive call handler must examine the flag appropriate to the process making the request and to the service being requested. Only if this flag is set may the service be performed. The set of flags associated with each process is part of its security sphere.

2.2. Supervisors.

A process at the root of a hierarchy of subprocesses is a "Supervisor" of those subprocesses. Supervisors accept messages (from handlers and other processes), interpret them, and control the computation of an appropriate response. Like all processes, Supervisors work in normal state and have free access to all resources defined for the security sphere in which they are initiated. The mapping from/

/from handlers to "top level" Supervisors is initialised when the OS is generated, but can be altered dynamically. This allows the installation management to shut down (or bring into service) the various subsystems controlled by the top-level Supervisors. (All other resources of the system are partitioned between subsystems in exactly the same way.)

3. Outline of a Specific Subsystem.

3.1 General.

This section sketches the features of a subsystem capable of supporting "batch" and multiaccess working. For brevity in the following we will call this subsystem SECANT.*

3.2 Structure of SECANT.

The hierarchy of processes which form SECANT are under the control of the top-level Supervisor. This process has three main functions.

(a) it schedules its subprocesses.

(b) it initiates new jobs.

(c) it provides commonly-required services to other components of MASS.

A new job is initiated in response to a message from some other process or from a handler. This message indicates where the commands defining the job will be found (e.g. in a file, a card reader, a teletype etc.). The Supervisor responds by creating a subprocess to act as the job organiser for that job. Each job organiser (JO) process reads commands from the designated source, interprets them and causes their execution.

Commands fall into several categories.

(a) Those commands which are executed directly by the JO itself we shall call metacommands. Metacommands have an effect on the running of the job itself. Examples are the "login" and "logout" commands.

(b) Those commands which are executed by a subprocess initiated by the JO we shall call, simple, commands. The code-part of the subprocess which the JO initiates will be an appropriate command program.

(c) Those commands which are executed by creating a text file through parameter substitution (and nesting the command source down a level to this file) we shall call macro-commands. Macro-commands expand to a series of commands and/or metacommands.

Thus/

*This is not an acronym!

/Thus SECANT is founded on just three fundamental mechanisms:

(a) the capacity of JO to execute a (small) set of metacommands;

(b) the ability to create a subprocess from a designated command program; and

(c) macro-expansion of command files.

This simplicity of structure allows a very inexpensive initial implementation, and yet contains the "hooks" to add further capabilities in response to user demand. Further, if a user installation finds the standard facilities inappropriate, he can easily supplant and/or supplement them with facilities of his own design.

3.3 The Command System.

3.3.1 General.

Commands are read and interpreted by the user's JO process. The commands defining a job form a command stream which issues from a number of command sources. A command source might be any of (a) an interactive console, (b) a paper tape or punched card reader, (c) the operator's monitor console, (d) a file on backing store, or (e) a running process. Execution of a command, or some other event, may cause the command source to change from one entity to another. Further, this change can be either temporary or permanent. In the former case the command stream is said to have nested down to a deeper command level. When the command stream reverts back to its previous source it is said to have nested up to a higher command level. The top level always corresponds to the command source which issued the "login". There is no ad hoc restriction on the deepest level that may be reached (e.g. by recursive invocation of a command file).

3.3.2 The Metacommands.

The metacommands provide the user with the following essential facilities:

(a) logging-in to the system;

(b) logging-out and ending the job;

(c) establishing conditions for an interruption of command processing to occur (c.f. ON statements in PL/I);

(d) /

(d) testing current conditions and proceeding accordingly;

(e) skipping forwards and backwards in the command stream;

(f) reverting to the previous command source with/without a failure indication;

(g) forcing an interruption of command processing (a break-in);

(h) returning from such an interruption and continuing command processing;

(i) abandoning all command processing beneath a specified level and continuing from that level;

(j) suspending an interrupted subprocess and recording its status in a file;

(k) re-establishing an interrupted subprocess from a file and restarting it;

(l) terminating the current job, interrupting another specified job, and making the command stream of the latter nest down to the current command source;

(m) signalling the end of an embedded data file in the job stream;

(n) declaring a set of variables whose scope is the current command source;

(o) assigning expressions to these variables.

3.3.3 Program-defined Commands.

The minimum useful set of program-defined commands will include at least the following:

(a) commands to communicate with the filing system in order to CREATE, DESTROY, RENAME, PERMIT (access to) and RETRIEVE (archived or corrupted) files;

(b) two editing commands (say EDIT and AMEND) permitting line-number and context editing respectively;

(c)/

(c) a command to insert a job in the Long-term scheduler queues (the LTS is a program that runs periodically and notifies the Supervisor of new "batch" jobs it should initiate);

(d) a command to define a macrocommand file;

(e) a command to define a new account;

(f) a command to define a new user;

(g) a command to "build in" a new command to the JO program.

Further commands will assist the mid-term scheduler (the MTS is a module of the Supervisor which allocates resources to the currently-active processes) by, for example, setting time and storage limits.

3.3.4 Command Formats.

The following suggestions seem adequate.

(a) At most one command can be present per source record.

(b) Long commands are continued over several source records by an appropriate continuation convention.

(c) Non-significant commentary (starting with a warning character) can be appended to any command.

(d) Blanks occur as significant characters only in quoted strings.

(e) Blanks may occur as non-significant "noise" characters only before/after delimiters.

(f) A character (which should be changeable by the user) must appear at the start of every command to distinguish it clearly from data embedded in the command stream. (We use " ! " in the examples below.)

(g)/

(g) Two such warning characters must appear at the start of every non-builtin command (e.g. " !! ").

(h) The general command format is

[!] ! [label:] commandname [parameterlist]

where the brackets enclose optional components. The parameterlist (if supplied) consists of a series of parameters, each separated from the preceding item by a comma.
e.g.

```
I  IF, ¬ MOUNTED (AFILE), TYPE
       "PLEASE MOUNT AFILE"

!  RENAME,  AFILE,  BFILE

!! SOLVEPDES,  BFILE,  RESULTSFILE

!  LAB3 :  PRINT,  RESULTSFILE
```

(i) Interactive users should enter commands via a formatter/prompter program which recognises a certain non-printing character as a parameter delimiter and prompt request. On recognising this character the prompter inserts a comma in the input buffer and prints an appropriate prompt.

(j) Actual parameters to commands should take on an appropriate syntax and be either a quoted string, a filename, an integer, ... etc, or an expression which evaluates to one of these.

(k) Parameters are called by value-result.

(l) It should be possible to compile heavily-used macros to equivalent command programs, so avoiding the overhead of macro-expansion on each call.

3.3.5 Bootstrapping./

3.3.5 Bootstrapping.

Note that a limited but useful system can be built by omitting the macro facility, all but the most basic metacommands, and most of the user-oriented program-defined commands. Using this system as a major tool further facilities can be added incrementally and in order of priority until the full specification above is met.

Such a "bootstrapped" implementation permits early release of the basic software and minimises the cost of extensions.

3.4 Input/Output.

Since it is not appropriate for processes working in SECANT to nominate specific physical devices in transfer requests, there must be a mapping to convert the virtual devices quoted by such processes to the actual devices on which the I/O operation finally occurs. Only rarely will an ordinary user process be permitted access to "basic" peripherals in this way. More normally, the virtual device will map on to some file in the SECANT filestore.

Significant advantages are gained when the mapping is decomposed into several stages. The greatest of these advantages is that the filing system can be interposed before the final stages, thus enjoying the use of the facilities they provide. In fact, it turns out that seven distinct levels can be distinguished. These are:

(a) the user level.

(b) the data base management system level,

(c) the "access method" level,

(d) the VIRTUAL I/O System, VIOS,

(e) the filing system,

(f) the LOGICAL IOCS, LIOCS, and

(g) the PHYSICAL IOCS, PIOCS.

3.4.1 VIOS.

VIOS maps virtual devices on to files or logical devices, using the identity of the requesting process and a mapping table for each process (which is set up by "open device" messages from that process).

3.4.2/

3.4.2 The Filing System.

The filing system is described more fully in section 3.5; here we only point out that its main function is to map filenames to logical devices, using the catalogued information in the filestore.

3.4.3 LIOCS.

LIOCS maps logical devices, which are specified as "the nth device of class m" to physical devices (the actual hardware of the installation). It achieves this by means of device availability tables and with the assistance of the operator.

3.4.4 PIOCS.

PIOCS is, in fact, the collection of device handlers which have been included in the Executive. PIOCS vets and implements requests from LIOCS. This involves -

(a) constructing appropriate IOP (channel) programs,

(b) ensuring that the I/O capacity of the system is not exceeded,

(c) reporting errors, terminations and other status changes to LIOCS,

(d) scheduling transfers on appropriate devices (e.g. discs and drums).

3.4.5 The I/O Hierarchy.

"FIELDS"

"RECORDS"

"RECORDS"

"BLOCKS"

"VIRTUAL UNIT"

"FILE" OR
"LOGICAL UNIT"

"FILE NAME"

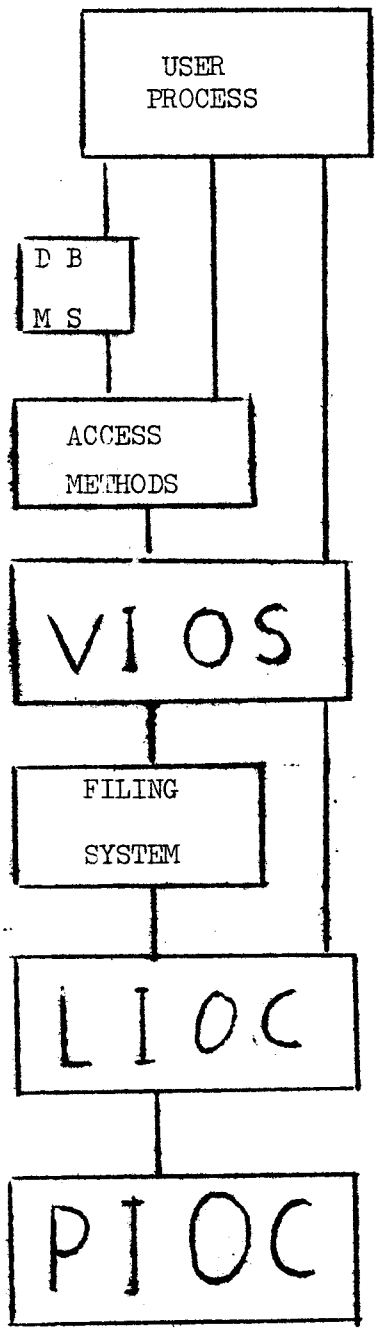
"LOGICAL UNIT"

"LOGICAL UNIT"

"PHYSICAL UNIT"

"PHYSICAL UNIT"

"IOP PROGRAMS,
INTERRUPTS,
STATUS FLAGS,
ETC."



4. The Filing System.

4.1 General.

The Filing System implements a filestore in backing storage volumes allocated to it by the installation management. The filestore is structured as a rooted tree whose depth is limited to 2. Nonterminal files in this tree act as catalogs. The storage structure chosen for all files is that of a randomly-addressable virtual address space implemented by means of a block of pointers to allocation units. By suitably choosing the size of the allocation unit the user can ensure efficiency of access for his application.

4.2 Catalogs.

Each catalog contains one entry per subordinate file. Each entry indicates:

(a) the file name, including its type;

(b) the account being charged for the space it occupies;

(c) the maximum and current number of generations;

(d) the physical position of the file (given as a volume serial number and the position of its descriptor on that volume's VTOC); and

(e) access control information for all classes of user.

4.3 Access Control.

Each file's catalog entry summarises the restrictions on access to the file for each class of user of the file. The possible classes are:

(a) the file's owner (i.e. the user who created the file);

(b) partners of the owner (i.e. all users charging their job to the account paying for the file);

(c) specific, named, users; and

(d) the "public", i.e. everybody else.

It will probably be most economical to place complete access control information in the catalog for classes (a), (b) and (d) and to record also the freest access permitted to any user in class (c). The record of access permission codes for users in class (c) will be completely set out only in the descriptor. Thus,/

/Thus, the most common cases can be decided without reference to the file itself (which may be on a demounted volume), yet the catalog entry is not expanded by (potentially very long) user/permission lists.

The modes of access which can be specified are any combination of

(a) execute only;

(b) read only;

(c) append only;

(d) update only (i.e. lock, read/write when locked, and unlock),

(e) write, read and append;

(f) status (e.g. delete, rename, etc) write, append, read.

4.4 Descriptors.

The catalog entry of a file points to its VTOC entry, which contains the file's descriptor. The descriptor includes:

(a) the date and time at which the file was created;

(b) the expiry date;

(c) the current and maximum sizes of the file;

(d) a list of all extents (or tape reels) occupied by the file;

(e) data for the dumping/archiving system, including statistics on the usage of the files;

(f) a field for the use of access methods and VIOS; and

(g) a list of all nominated users of the file with their access permission codes.

4.5 Dumping/Archiving.

A dumping/archiving system similar to that of the Titan Multi-access System should be implemented. This can be done by a user program run on behalf of an appropriately privileged user (e.g. the system manager).

5. User and Resource Control, Accounting.

5.1 General.

The user/resource control and account facilities of SECANT should allow as close a degree of control over the use of resources and the dispensation of privilege as is desired by the installation. In particular, it is possible to create accounts and sub-accounts to any level and allocate resources to them, and to create users and sub-users to any level specifying which accounts each user may charge his work to and the limits on his expenditure.

5.2 User Profiles.

The user control modules maintain a file with an entry for each authorised user of the system. This entry specifies:

(a) the user's (unique) name;

(b) his password (suitably enciphered);

(c) the name of his immediate superior (i.e. the user who created this entry);

(d) the names of his immediate inferiors;

(e) permissions specifying which classes of built-in command he may use;

(f) statistics on his usage of the system;

(g) for each and every account to which he can charge his work, a list giving his current and maximum expenditures for the various resources owned by that account.

5.3 Account Profiles.

The accounting modules maintain a file with an entry for each account in the system. This entry specifies:

(a) the (unique) name of the account;

(b) the name of the immediately superior account;

(c)/

/(c) the names of the immediately inferior accounts;

(d) the names of all authorised users of the account;

(e) a list of the resources available to and currently consumed by the users of the account.

Creating a subaccount and allocating resources to it diminishes the resources of the parent account accordingly. However, all users of an account "pool" their resources - only their limits of expenditure are recorded in their individual profiles. Thus the undesirable situations that can arise when resources are allocated directly to users are avoided.

R E F E R E N C E S

1. Dijkstra, E.W.

   The structure of the "THE" multiprogramming system.

   Communications of the ACM, 11, 341-346 (1968).

2. Hansen, P.B.

   The nucleus of a multiprogramming system.

   C.A.C.M. 13, 238-241 (1970).

3. Madnick, S.E.

   Design strategies for file systems.

   MAC TR-78.

4. Spooner, C.R.

   A Software Architecture for the 70's, Part 1.

   Software Practice and Experience Vol. 1, No. 1, 5-37 (1971).

5. Introduction to the Virtual I/O System, University of Glasgow, Computing Dept., MULTUM Operating System Memo No. 5, (January, 1973).